第1章 システムLSI構築のかなめ，トップダウン設計・検証

——大規模FPGAとCPUコアを活用して効率化

大庭信之

大規模LSIの開発では，はじめからすべてをRTL（register transfer level）で設計することは容易ではない．抽象度の高いレベルからトップダウンに設計を進めるのが一般的である．ここでは，設計をいくつかの階層に分けて考え，トップダウンでシステムLSIを設計する手法を解説する．また，抽象度の高い段階や，複数の抽象度が混在する設計レベルで検証を行うために，FPGAとCPUコアを活用する手法について説明する．

（編集部）

近年，半導体技術の進歩とともに，より多くの機能が1個のLSIへ集積できるようになってきました．さまざまな回路ブロックとマイクロプロセッサを混載したシステムLSI（SOC：system on a chipと呼ばれることもある）が注目されています．

LSIの規模が大きくなり，いろいろな機能が集積されるに従って，その設計と検証には多くの時間と労力が必要になります．そこで，大規模LSIの設計では，最初からいきなりRTL（register transfer level）コードを記述するのではなく，高位のC++モデルなどを作ることで，トップダウンの設計手順を踏むことが多くなっています(1)．また，検証作業もRTLコードのみを使うのではなく，より高位のモデルを使って高速に行うことが多くなりました．しかし，依然としてLSIの開発期間の7割が検証に割かれているという報告もあり(2)，多くの時間と労力が必要です．

検証作業を効率化・高速化する一つの方法として，FPGAなどを用いたハードウェア・プロトタイピングがあります．LSIを製造する前に，設計をFPGA上に実装して検証する方法で，今では広く用いられています(2),(3)．

FPGAの進歩も著しく，大規模化，高速化だけでなく，いろいろなハードIP（intellectual property）を搭載したシリーズも登場してきています．

話が少々それますが，時代ごとにLSI技術の進歩をけん引した電子部品を「テクノロジ・ドライバ」と呼ぶことがあります．1980年代はDRAM，1990年代はマイクロプロセッサ，そして2000年から現在に至るまではFPGAがそのテクノロジ・ドライバであると言われています．大容量かつ高速なFPGAが入手できるようになり，通信機器，ネットワーク機器，ディジタル家電など，いろいろな分野で活用されています．

FPGAベンダの米国Xilinx社からは，PowerPCのハード・マクロを搭載するFPGA「Virtex-II Pro」，「Virtex-4」シリーズが発売されています．32ビットRISCプロセッサPowerPC 405コアが1個ないし2個搭載され，最大450 MHzで動作し，700Drystone MIPSを超える性能を発揮します．

本稿では，トップダウン型のLSI設計について，実例を交えながら述べます．例えば，FPGAに内蔵されたマイクロプロセッサを用いて，複数の設計抽象度で記述されたLSIを検証する方法を紹介します．

1．トップダウン設計・検証

LSIの大規模化，高機能化に伴い，全体を効率良く設計・検証する方法が求められています．そこで，近年，システム・レベルからネットリスト・レベルまで一貫性のあるトップダウン設計手法が用いられるようになってきました．

● トップダウン設計とは

最初にLSIの仕様を仕様レベルで記述します．端的に言えば，LSIが何をするのかを明確にします．続いて，設計

表2 LSI設計のポイントの例

ポイント	説 明
ソフトウェアとハードウェアのすみわけ	一つの処理を実行するため，1）すべてをハードウェアで行う，2）汎用CPUもしくはDSP上で走るソフトウェアで行う，3）あるいはハードウェアとソフトウェア両方で協調しながら行う，という3通りの場合が考えられる．
ハードウェア化の効果，ソフトウェアで実現するメリット	一般的に，ハードウェア化することで処理速度は上がり，また消費電力は下がる（そうならないとハードウェア化する意味がない）．しかし，ハードウェアでぎちぎちに組んでしまうと，アルゴリズムやデータ構造を少し変えようと思っても対応しにくくなる．一方，ソフトウェアで実現すれば，あとからアルゴリズムやデータ構造の変更要求に柔軟に対応できる．
レジスタ，メモリ	シングルポートかマルチポートか．同時に二つ以上のアクセスが発生する場合，マルチポートのレジスタやメモリを用意するのが簡単．しかし，デュアルポート・メモリは，シングルポート・メモリに比べて，ハードウェア量が50％～100％増しになることがある．また，デュアルポート・メモリであっても，同時に同じアドレスにライトとリードが発生した場合，どのデータがリードされるかはメモリによるので，注意が必要．資源節約のため，一つのレジスタやメモリを複数の用途で使いまわすこともある．
タイミング（動作遅延）の見積もり	言うまでもないことだが，所定の速度（周波数）で回路が動作しなければならない．例えば加算器でも，ハードウェア量が少ないが遅いリプル・キャリ型を用いるのか，ハードウェア量が多いが速いキャリ・ルックアヘッド型を用いるのか，というように，同じ処理でも異なった回路構成にできないかを考える．
マルチサイクル動作	通常の同期システムでは，グローバル・クロックに同期した入力ラッチと出力ラッチにはさまれた組み合わせ回路は，1クロック・サイクル中に演算を終了している必要がある．しかし，ものによっては2クロック・サイクルかかってよいケースもある．あえてメイン・クロックの半分の周波数をもつクロックを別に用意し，回路の最適化を図ることもある．
レイテンシとスループット	レイテンシは入力を与えてから出力結果が得られるまでの時間のことであり，スループットは単位時間当たりに得られる結果量である．パイプラインを深くすることでスループットを上げるという手段はよく用いられるが，そうするとレイテンシが長くなる．概してレイテンシが長い回路は設計がむずかしくなる．とくに，データによってレイテンシが変わってくるような回路は，動作をきちんと把握しないと設計が破たんしてしまう．

をより詳細なレベル，すなわちトランザクション・レベル，サイクル精度レベルへブレークダウンしながら，論理回路を合成可能なRTLまで変換していきます（**表1**）．もちろん，かならずしもこのステップを1段1段たどるとはかぎりません．例えば，簡単なハードウェア・モジュールであれば，仕様レベルから直接RTL設計へ進むこともあります．

求められている処理，機能をハードウェア化するためには，アルゴリズムを正確にハードウェアで実現するだけでなく，いろいろな条件や制約をクリアしなければなりません．そのためには，**表2**のような観点からハードウェアを煮詰める必要があります．

トップダウン設計を行うことにより，このような条件，制約を的確に設計へ取り込みやすくなります．そして，複雑な大規模LSIでも見通し良く，バグも少なく，開発することができます．また，より高位のレベルで記述されたモデルを使うことにより，すべてのモデルをRTLで記述した場合と比べて，高速にシミュレーションすることができます．

トップダウン設計・検証は，具体的にどのように進んでいくのでしょうか．

図1はシステムLSIの一例です．このシステムLSIにはPowerPC 405がCPUコアとして搭載され，システム・バス（PLB：Processor Local Bus）を通して，メモリ・インターフェース，MPEGエンコーダ，MPEGデコーダ，DES（data encryption standard）/AES（advanced encryption standard）暗号エンジンと接続されています．また，PLBはPLB-OPBブリッジを介してI/Oバス（OPB：On-chip Peripheral Bus）とつながります．OPB上にはRSA（Rivest, Shamir and Adleman）/ECC（elliptic-curve cryptography）暗号エンジン，乱数発生回路，汎用I/Oインターフェースが接続されています．では，どのように設計していくのか，レベルごとに見ていきましょう．

表1 設計抽象度
下位にいくほど，より最終回路に近くなっていく．動作タイミングも正確に記述される．

設計抽象度	検証レベル	タイミング	実 装
仕様レベル	コンセプト	untimed	ソフトウェア
トランザクション・レベル	アーキテクチャ	timed/untimed	ソフトウェア
サイクル精度レベル	マイクロアーキテクチャ	timed	ソフトウェア
RTL（レジスタ・トランスファ・レベル）	実 装	timed	ハードウェア

● 仕様レベル（specification level）

仕様レベルでは，LSIのコンセプトおよび何を行うものであるのかを記述します．

「本LSIは暗号機能を搭載した画像処理プロセッサです．画像処理はMPEG-4規格をベースに行います．画像データの暗号化/復号化を含んだセキュアな伝送はSSL（secure socket layer）プロトコルを使います」．

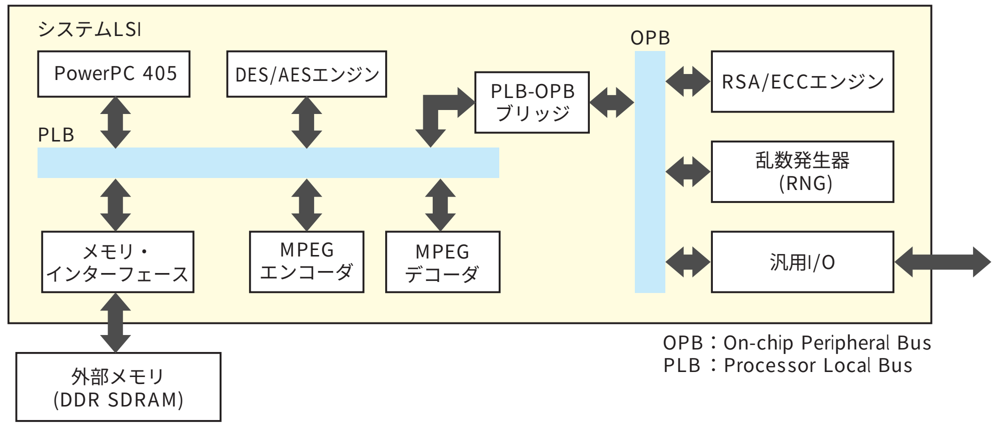


図1
暗号回路を搭載した画像処理LSIの設計例
PowerPC 405をCPUコアとして用い，MPEGエンコーダ，MPEGデコーダ，DES/AES共通かぎ暗号エンジン，RSA/ECC公開かぎ暗号エンジン，乱数発生器を搭載する．

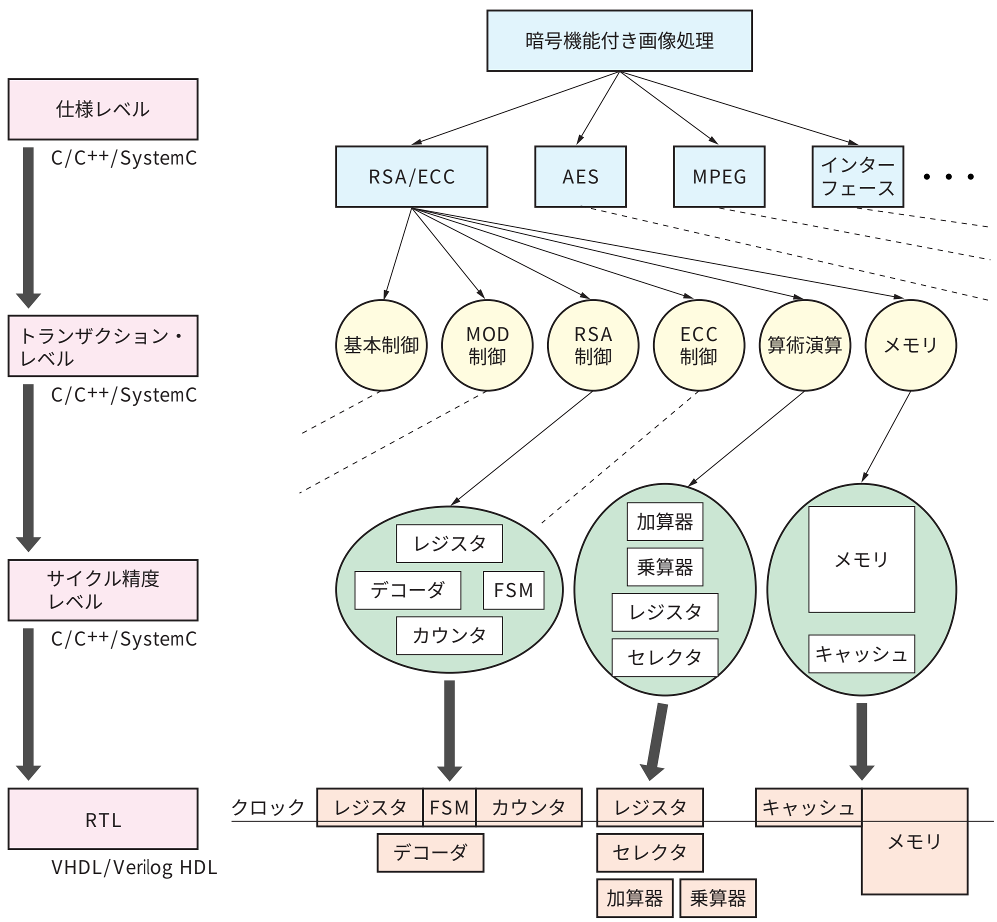


図2
設計のブレークダウンの例
RSA/ECCエンジンがレジスタ，ステート・マシン，カウンタへ実装されている過程がわかる．

このようなぐあいに，LSIの機能をC++で記述します．今回は要求された画像処理，暗号処理のスループット性能がPowerPC 405を使ったファームウェアだけでは達成できないレベルであったため，図1に示すように専用のハードウェア・モジュールを設計し，統合しました．

また，暗号に関連する演算はセキュリティ上の配慮から，なるべくハードウェア・モジュール内で閉じて行いたいという希望もあります．このレベルで一連の処理手順をC++で記述しておくと，後々の検証時に見通しの良いデバッグが行えます．例えば，画像データ暗号化のための秘密かぎ生成処理では，

1）乱数発生器で乱数を発生させる．
2）DESエンジンで発生した乱数をスムージングし，秘密かぎを生成する．

リスト1　仕様レベルとトランザクション・レベルのC++ コーディング例

仕様レベルでは普通のC++ プログラムとほとんど同じ．トランザクション・レベルではカウンタのリセット，増減などのハードウェアを意識したコードになっている．

```
class mod_exp_hm
{
  public:
//for event-drive/clock-driven modeling
  int issue();
:    private:   int i, j;             :  };
```

```
int mod_exp_hm::issue()
{
    :
    for (i = core_bitwidth-1; i > 0; i--){
    :
       for(j=bitwidth-1; j>0; j--) {
                :
       }
    }
}
```

(**a**) 仕様レベル

```
class coutner_class  {
public:
 void issue(void);
 bool iszero();
 void dec();
 void set(int calue);
 coutner_class();
 virtual ~coutner_class();
private:
 int counter;
 int in_dec;
 int in_set;
};
void coutner_class::set(int value)  {
 in_set = value;         // dec() flag
}
void coutner_class::dec()  {
 in_dec = 1;    // dec() flag
}
bool coutner_class::iszero()  {
 if(counter == 0) {
  return true;
 } else {
  return false;
 }
}
void coutner_class::issue()  {
 if(in_set > 0) {
  counter = in_set;
 } else if(in_dec > 0) {
  counter = counter - 1;
 }
 in_set = -1; // reset set() flag
 in_dec = -1; // reset dec() flag
}
```

```
class mod_exp_hm
{
  public:
// for event-drive/clock-driven modeling
  int issue();
   :
  private:
  counter_class i, j;    // use counter class
   :
 };



Int mod_exp_hm::issue()
{
  :
   i.set(core_bitwidth.minus_one());
   i.issue();
   do {
  :
     j.set(core_bitwidth.out());
     j.issue();
     do {
            :
     j.dec();
     j.issue();
     } while(j.iszero()==false);
   i.dec();
   i.issue();
   } while(i.iszero==false);
}
```

(**b**) トランザクション・レベル

3）RSAエンジンを使い，秘密かぎを公開かぎで暗号化する．

という手順を踏みます．また，一連の処理でどこが律速段階なのか，どの部分をハードウェア化し，そして並列処理することで高速化できるのかを把握できます．

ここから図2に示すように，合成可能なRTLへブレークダウンしていきます．最初にRSA，ECC演算は普通のC++ プログラムとして記述し，テスト・データを与えて正しく動作するかどうかを検証します．これをいきなりRTLへ落とすことは難しいので，まずトランザクション・レベルへ変換します．

● トランザクション・レベル(transaction level)

LSIの内部構成を具体化します．

トランザクション・レベルでは，バス上でどのようなデータ転送が発生するかを正確に記述し，把握しながら，全体構成を決定します．正しく動くことを確かめることはもちろん重要ですが，バスのバンド幅に不足がないかどうかの性能評価見積もりも行います．

画像処理やDES/AESの秘密かぎ暗号処理は，データ転送のバンド幅が高いため，PLB接続としました．一方，RSA/ECC暗号処理は，一つのデータ・サイズは大きいのですが(RSAで1024ないし2048ビット)，セキュア・セッ

ションを最初に構築するときにしか用いられません．また，1回の演算に100万サイクル以上の長い時間がかかるので，データ転送のバンド幅は低いものです．したがって，OPBへ接続することとします．乱数発生器も頻繁に使うものではないので，OPB接続とします．同じく，汎用I/Oインターフェースも動作速度が遅いので，OPBへ接続します．

リスト1に，仕様レベルとトランザクション・レベルのC++で記述した設計例を示します．クラス中のプライベート変数がハードウェア実体のある変数に変換されています．また，`for`ループがカウンタ・クラスに変換されています．データ入出力操作はメソッド呼び出しで実現されています．プライベート変数は，最終的にRTLですべてレジスタ記述に変換されることになります．

● サイクル精度レベル(cycle accurate level)

トランザクション・レベルではイベント・ドリブンで動作していました．例えば，バス上にリード・リクエストというイベントを発生し，そのアクノリッジというイベントを待つというハンドシェイクです．それに対してサイクル精度レベルでは，すべての機能モジュールや論理は，LSIの内部クロックに同期して動作が進んでいきます．各機能モジュール中のレジスタ，デコーダ，カウンタ，ステート・マシンもクロック・サイクルごとにどういう動作を行うかが正確に記述されます．

今までC++中に書かれていた条件分岐，すなわち`if`文はハードウェアへ直接実装可能な記述に変換します．メモリも実際のLSI中で使える構成(ビット幅，ライン数，ポート数など)を考慮してメモリ，あるいはキャッシュ・メモリへ変換して実装します．

一般に，性能とゲート数，動作周波数，消費電力は比例関係にあります．与えられた仕様を満たし，かつゲート数や消費電力が最適なポイントを見つけるのも大事な作業です(**図3**)．

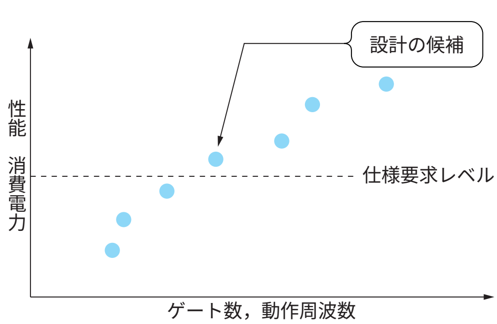


図3　性能とゲート数，消費電力のトレードオフ
与えられた仕様を満足する最適なポイントを見つける必要がある．

● RTL

RTLは，そのままLSIやFPGA向けに合成・実装できるレベルです．

2. FPGAとCPUコアを用いた設計・検証

近年のLSI開発でもっともタフなフェーズは検証です．本LSIの場合であれば，以下に述べる理由から，ソフトウェア・シミュレーションだけでたくさんのテスト・ケースを流すことはかなり難しいことがわかります．

公開かぎ暗号(RSA，ECC)の演算には膨大なサイクル数を必要とします．例えば，2048ビットRSAの演算では1億クロック・サイクル以上が必要です．これをソフトウェア・シミュレーションで流すと，一つのデータを計算するだけでも数日かかってしまいます．シミュレーション中に問題が起きた場合，バグの原因をトレースするためにキーとなる信号の中間状態をすべて記録しておく必要があります．信号状態を長時間ハード・ディスクに記録するため，大きな容量のハード・ディスクを用意するか，または記録する信号線の本数を減らさなければなりません．あとから，「あの信号も見ておけばよかった」ということがあります．また，シミュレータのライセンス形態によっては，一つのシミュレーションを流している間，ライセンスを占有するため，ほかのシミュレーションを並行して流せないということもあります．

AES，RSA，ECCはガロア体をベースにした演算であり，比較的複雑な演算処理を必要とします．一気にハードウェアへ落とすことは容易ではありません．通常，一つの暗号処理は複数の演算からなっています．例えばSSLでは数回の公開かぎ暗号，秘密かぎ暗号の演算を必要とします．これら一連の演算をすべてシミュレーションするには数十億クロック・サイクル以上が必要です．

● FPGAによるプロトタイピングの効果

RTLのモデルを高速に検証するために，FPGAを活用したハードウェア・プロトタイピングが広く用いられています．ターゲットの設計をすべてFPGAに実装できれば，高

表3 RSA演算器（RTL記述）の検証にかかる時間

処　理	サイクル数	ソフトウェア・シミュレーション	FPGAエミュレーション
1024ビットRSA 32ビット演算コア	400万	6～7時間	0.3秒
2048ビットRSA 32ビット演算コア	3200万	2～3日	3秒
1024ビットRSA 16ビット演算コア	1600万	1日	1.5秒
2048ビットRSA 16ビット演算コア	10200万	1週間	10秒

注：シミュレータはMentor Graphics社のModelSim SEバージョン5.4で，2GHz動作のPentium 4と1Gバイトのメモリを搭載したワークステーション（IBM社のIntellistation MPRO）で動作させた．FPGAは33MHzのシステム・クロックで動作するXilinx社のFPGA「XC2VP20」．

速で正確な検証が行えます．たいへん効率の良い検証手法といえます．筆者らの経験では，ソフトウェア・シミュレーションに比べて，3～6けたほど速い速度でテストすることができるようです．

RSA公開かぎ暗号演算はたいへん時間がかかると述べました．実際には**表3**に示したような検証実行時間がかかります．使用シミュレータは，米国Mentor Graphics社のModelSim SEバージョン5.4です．ここで用いたシミュレーション環境は，2GHz動作のPentium 4と1Gバイトのメモリを搭載したワークステーション（米国IBM社のIntellistation MPRO）と，33MHzのシステム・クロックで動作するXilinx社のFPGA「XC2VP20」です．所要時間を大ざっぱに比較してみると，FPGAエミュレーションはソフトウェア・シミュレーションの10万倍高速だということがわかります．

● FPGAによるプロトタイピングの問題点

しかし，ハードウェア・プロトタイピングも万能ではありません．CPUコアを内蔵するシステムLSIの場合，一つのLSIの内部は，通常，複数の機能モジュールをバスで接続する構成になっています．それぞれの機能モジュールが，開発途中の段階で異なった設計抽象度のデータであることがよくあります．

簡単なモジュールならすぐにRTLに落とせるでしょう．しかし，複雑で大きなモジュールはRTLに落とすまでに時間がかかります．また，外部からIPコアとして機能モジュールを調達した場合，RTLあるいはネットリスト・レベルのデータしか入手できないこともあります．逆に，ある特定の半導体製造プロセスにしかマップできないIPコアで，FPGAへ実装できず，検証には別のビヘイビア・モデルを使わなければならないこともあります．

すなわち，LSIの設計開発途中では，**表1**に示すいろいろなレベルの機能モジュールが混在した状況で検証を行わなければならないことがあります．本稿ではこのように複数のレベルのモジュール（モデル）を接続していっしょに検証する方式を「ミックス・レベル動作検証」と呼ぶことにします．

このようなミックス・レベル動作検証をハードウェア・プロトタイピングで行う場合はどうしたらよいのでしょうか．

従来は，RTLで記述された合成可能なモジュールだけをFPGAへ実装し，それ以外のC++やビヘイビアなどのより抽象度の高いレベルで記述されたモジュールはホスト・パソコンで実行し，それらをPCIバスやPCI-Xバスなどを介して仮想的なバスを構築し相互接続することで，検証を行っていました（**図4**）．

現在，CPUの動作速度が3GHzを超えるホスト・パソコンも簡単に入手できるようになり，かなり高速なシミュレーションが可能となっています．また，FPGAも高速化，大規模化が進み，ターゲットのLSIと同等の速度（リアルタイム）で動かせる場合もあります．しかし，パソコンと

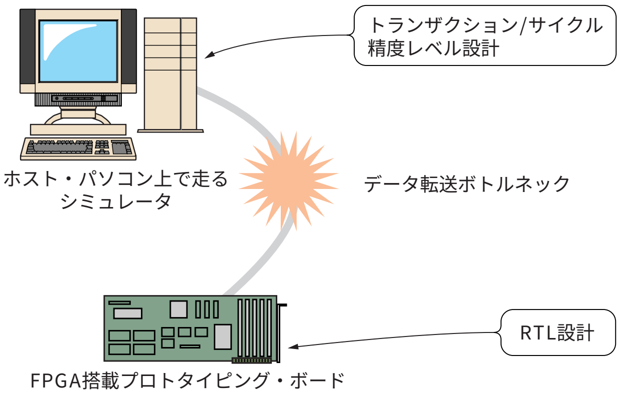


図4　従来のプロトタイピング・ボードとホスト・パソコンを協調させたミックス・レベル検証

FPGAへ直接マッピングできる合成可能な設計はプロトタイピング・ボードへ実装する．トランザクション・レベル（ビヘイビア・レベル）設計はホスト・パソコン上でシミュレータを使って実行する．ホスト・パソコンとプロトタイピング・ボードの間のデータ転送がボトルネックとなって，全体の検証速度の低下を招くことが多い．

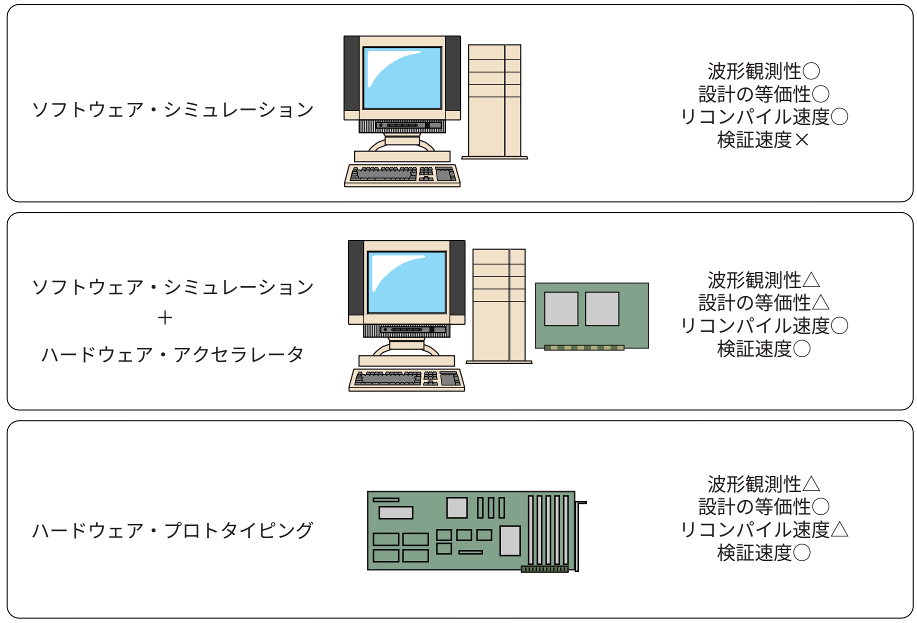


図5
代表的な検証方法の比較
それぞれの特徴を理解して使い分けることで効率の良い検証が行える．

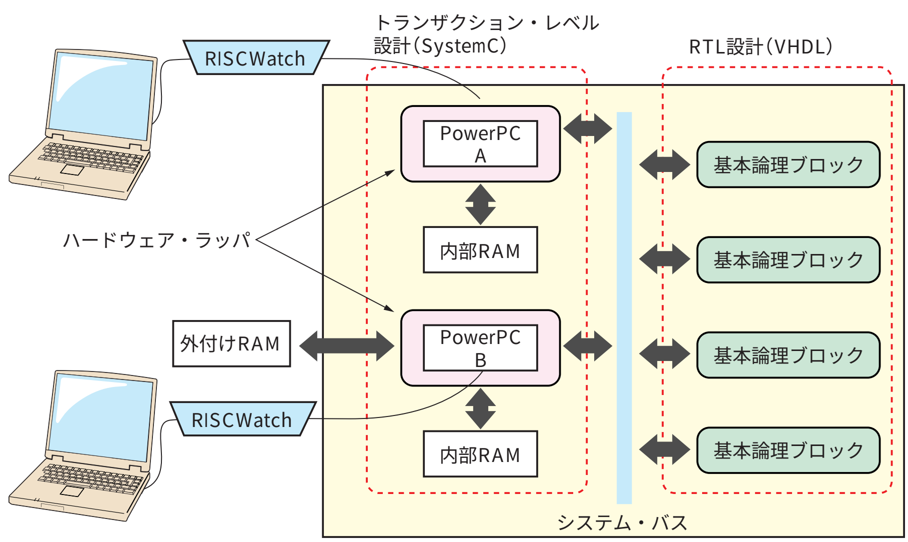


図6
FPGAを用いたミックス・レベル検証
トランザクション・レベル設計をPowerPCでシミュレーションし，RTL設計はそのままFPGAの基本論理ブロックへマッピングする．2個のPowerPCにはそれぞれRISCWatchをつなげて，プログラムのロード，開始，ブレークなどのデバッグを行う．

FPGAの間のデータ転送がボトルネックとなって，全体の性能が上がらないことがあります(**図4**)．

実際には，ソフトウェア・シミュレーションやFPGAプロトタイピングを組み合わせて，いろいろな観点，状態から，なるべく多くのテストを行い，検証のカバレッジ(網羅率)を上げることが重要です．**図5**に代表的なLSI検証方法についてまとめてみました．

● ミックス・レベル動作検証環境

CPUコアを含めてFPGAの機能を最大限利用することで，ミックス・レベル検証を行う手法を紹介します．この手法をひと言で言うと，RTLでFPAGへ合成マッピング可能なモジュールは従来どおりFPGAの基本論理ブロックへ実装し，FPGAへ直接マッピングできない高い抽象度で記述されたモジュールはFPGA内部のCPUコア上でソフトウェアとして実行する，というものです．この手法により，C++やSystemCで記述された，FPGAへ直接マッピングできないモジュールもFPGA上で実行，検証することができます．また，異なった抽象度の設計(モデル)をFPGA上で同時に走らせて検証することができます．

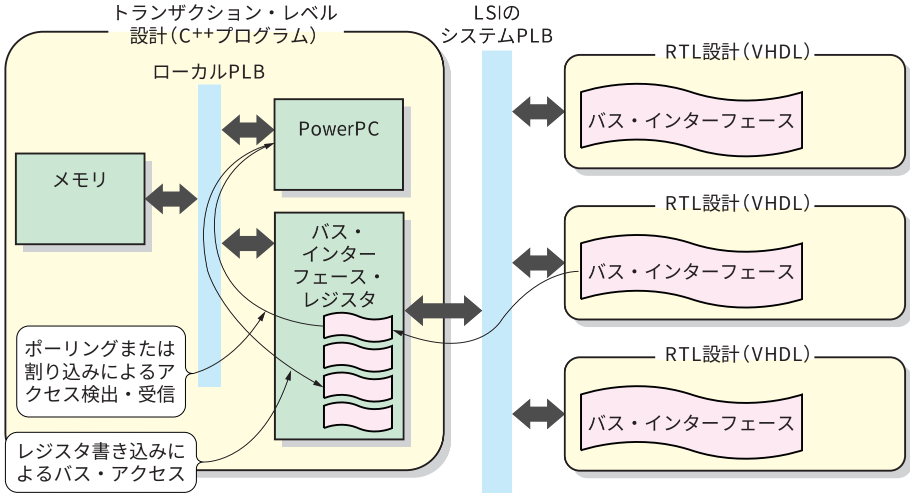


図7
PLBと接続するためのハードウェア・ラッパ
トランザクション・レベル設計のバス・アクセスは，PowerPCによるレジスタ・ライトで実装される．

図6に検証のためのFPGAマッピング例を示します．RTLで記述されたモデルは従来どおり合成され，FPGAの基本論理ブロックへ直接マッピングされます．一方，トランザクション・レベルのモデルはC++またはSystemCで記述され，二つのPowerPCの上でソフトウェアとして走ります．トランザクション・レベルで記述されたモジュールは，システム・バスを通してほかのモジュールと通信します．図に示した例で，PowerPC AはFPGAの内部メモリだけを使っています．一方，PowerPC Bは内部メモリのほかに外部メモリを持たせることで，より大きなモデルに対応できるようになっています．

二つのPowerPCにはそれぞれリアルタイムにデバッグを行えるインサーキット・デバッガ「RISCWatch」(4)をIEEE 1149.1(JTAG)インターフェース経由で接続しています．ハードウェア・プロトタイピングのかなめである実行速度をなるべく損なわないようにするため，PowerPC上でOSは走っていません．C++やSystemCで記述された設計がホスト・パソコン上でコンパイル・リンクされ，生成されたバイナリ(ロード・モジュール)を直接RISCWatch経由でPowerPCに接続されたメモリ上へダウンロードし，OSなしで実行しています．

また，検証担当者が容易にPLB，OPBと接続できるように，ハードウェア・ラッパを用意しました．ハードウェア・ラッパ中に埋め込まれたレジスタはPowerPCから直接アクセス可能になっており，それを使ってPLB，OPB経由でほかのモジュールとハンドシェイクを行ったり，データを送受したりできるようになっています．バスから来るアクセスはソフトウェア・イベントとしてポーリングまたは割り込みで検出していますが，割り込みはレジスタ退避などに時間がかかるため，おもにポーリングを用いています(図7)．バス上の信号などはRISCWatchから直接アクセス可能なレジスタの状態としてモニタすることができるので，効率の良いデバッグが行えます．

トップダウン設計例ではC++を用いましたが，筆者らはSystemCもPowerPC上で動かせる環境を作っています．SystemCには，

- datatypes(基本データ・タイプ)
- tracing(信号および変数のトレース)
- communication(モジュール間コミュニケーション)
- utils(キュー，リンク・リスト，ハッシュなどのデータ構造)
- qt(並列動作を実現するためのスレッド実装，Quick Threads)
- kernel(SystemCの最上位レベル実装)

の六つのクラス・モジュールがあります．今まで，x86系やSPARC系などのマイクロプロセッサには，これらが実装されています．datatypes，tracing，communicationの各クラスは容易に移植できますが，qtにはページ制御やスタック制御が入っており(5)，マイクロプロセッサ・アーキテクチャに依存する部分が大きいので，PowerPC用に作り直しています．qtにかかわるルーチンは一部をアセンブラで記述することで，SystemCの実行速度の向上を図っています．

3. まとめと今後の課題

ひと昔前のLSI検証といえば，VHDLやVerilog HDLでぎちぎちに書かれたRTLコードを長い時間かけてシミュレーションしていました．最近は，C++やSystemCで記述された抽象度の高い設計（モデル）を使うことで，システム全体のシミュレーションが高速に実行できるようになってきています．

しかし，ハードウェア・プロトタイピングによるシステム・テストの威力は健在です．ソフトウェア・シミュレーションが速くなったとはいえ，依然としてテスト速度がけた違いに速いということです．そしてFPGAの大規模化・高速化によってターゲットLSIをリアルタイムでエミュレーションすることも可能になってきたことも理由の一つです．

実際のOSやアプリケーションを流し，システム全体を検証することは，ソフトウェア・シミュレータ上でも不可能ではありませんが，ハードウェア・プロトタイピングを使うことでより現実の速度に近くになり，効率良くテストやデバッグを進めることができます．

従来は，LSIの内部コンポーネントがすべて合成可能なRTLで記述されていなければ，FPGAへ実装するハードウェア・プロトタイピングを使えませんでした．本稿で解説したように，FPGAとCPUコアを利用する手法をとることで，開発の早い段階，すなわち仕様レベルやトランザクション・レベルの設計を含めたミックス・レベルの検証をハードウェア・プロトタイピング上で行えるようになります．

筆者らは，Virtex-II Proを使っていたため，実は一つ問題がありました．CPUコアであるPowerPC 405は浮動小数点演算器を持っていません．したがって，C++/SystemCプログラムが浮動小数点演算を持っていると，ソフトウェアでエミュレーションする必要があります．PowerPC上でソフトウェアとして実行する仕様レベルやトランザクション・レベルの設計の動作速度が極端に遅くなってしまいました．

Xilinx社の最新のVirtex-4 FXシリーズでは，PowerPCのAPU（Auxiliary Processor Unit）インターフェースと呼ばれるコプロセッサ・バスがサポートされ，浮動小数点演算アクセラレータをPowerPCのそばに置けるようになりました．APUは，通常のプログラム・コードにAPU専用の命令を埋め込み，パイプライン・ストールのない自立型命令をサポートするものです．一つの命令で最大四つまでの32ビット・ワード・データ転送に対応します．APUインターフェース接続の浮動小数点演算器を使うことで，C++/SystemCプログラムが浮動小数点演算を使っていても，十分な速度が得られるものと考えています．

FPGAのマイクロプロセッサ・コアの応用の一例として，本稿が読者の皆様の参考になれば幸いです．

参考・引用*文献

(1) 特集1 SOC開発プロジェクトを成功させるには…，Design Wave Magazine，2002年8月号，pp.35-90.
(2) 大庭信之，森岡澄夫，高野光司；FPGAプロトタイピング環境を利用したハードウェア開発（前編），Design Wave Magazine，2003年2月号，pp.136-141.
(3) FPGA活用チュートリアル，CQ出版社，2004年3月.
(4) RISCWatch Debugger User's Manual，http://www-3.ibm.com/chips/techlib/techlib.nsf/products/RISCWatch_Debugger，2003年5月.
(5) David Keppel；"Tools and Techniques for Building Fast Portable Threads Packages"，*University of Washington Technical Report UWCSE 93-05-06*，1993年5月.
(6) Nobuyuki Ohba and Kohji Takano；"An SoC Design Methodology Using FPGAs and Embedded Microprocessors"，講演番号45-3，41st Design Automation Conference，2004年6月.

おおば・のぶゆき
日本アイ・ビー・エム（株）

<著者プロフィール>
大庭信之．日本IBM 東京基礎研究所勤務．計算機アーキテクチャに関する研究に従事．ooba@jp.ibm.com